FDTs foundations, methods and tools in the educational context: a DIT-UPM experience

T. Robles, J. Quemada, J. Salvachúa, A. Azcorra, D. Larrabeiti,
J. A. Mañas, T. De Miguel, S. Pavón, G. Huecas

DIT-UPM
Technical University of Madrid
Ciudad Universitaria s/n, 28040 Madrid SPAIN
trobles@dit.upm.es

Abstract

DIT-UPM has been involved from the beginning in the definition, development and application of Formal Description techniques (mainly LOTOS and ESTELLE). In an industrial and academically context that demands formation in such areas, DIT-UPM has been teaching courses about these items during the last years. This paper described the context, organization of the courses, examples and case studies used for illustrating concepts, tools and results of this experience.

1 Introduction

The definition of the ISO - OSI reference Model created the need of rigorous specification techniques to support an effective, sound and unambiguous definition of a complete architectural model, including many layers, protocols and services. Two competing Formal Description Techniques (FDTs), which later were called Estelle and LOTOS, started to be developed. The first one was based on an EFSM (Extended Finite State Machine) and the second one was based on a process algebra approach.

The definition of the whole architecture implied a big challenge which needed a technique providing a rigorous support to most (desirably all) tasks of a complex system design cycle. Protocol and service specifications were the first tasks undertaken, but these were not the only places were FDTs should play a role and provide a competitive advantage. Other important design activities were needed, validation and verification of the correctness of the protocol, performance analysis, conformance testing and implementation derivation.

All this process needs an accompanying educational effort to create the skills and expertise needed in a large industrial development. DIT-UPM made a large effort to support the development of LOTOS, which included contributions to the definition of the language, specification of protocols and services, tool development and large case studies. All this was complemented with the educational and training activities that led to successful application of LOTOS in an industrial environment.

Many results and conclusions were obtained in this process which are of general validity, despite the situation in which ISO promoted FDTs were left when the market chose the internet protocols for product development due to the existence of more mature implementations. This papers tries to summarize all what was learned during this process, as well as the main results and conclusions obtained.

The rest of the paper is organized as follows: Section 2 described the context where LOTOS courses where created and granted (industrial, research and academically considerations are taken into account;

Section 3 goes into the problem of generating didactic material for supporting the courses; Section 4 describes the structure and organization of a full course that combines theory and practices; Section 5 described theoretical part of the course and section 6 the practical session; and finally section 7 outlines the conclusions and learned lessons after some years of teaching courses on FDTs.

2 Promoting FDTs in context

Teaching something can not be considered in void, the context surroundings are really important. You must take into account the demand of the knowledge you want to teach by your environment.

In the framework of a technical school the driven force always is the demands of the companies around it. In the case of Technical University of Madrid, and specifically the Department of Telematic Systems engineering there is several focus that demands training in the FDT area.

The first one is the internal necessity of the University of Trained Research Personnel that will participate in the research activities of the University. We include into this group the courses that we have organized for another Spanish Universities as well the courses for research departments of companies. But this internal demand is not enough for justifying the development of a completely new set of material and courses. External demand is needed. In our case this demand comes from two different points: local industries wants to promote the skill of their personnel in those areas, with the intention of using those FDTs in the development and production of new products. That is the case of companies such Ericson, Alcatel, Telefonica or Indra.

A second group of companies especially actives in the use of these technologies are the companies involved in the development of product and technologies for critical applications. In our case this work was mainly centered on European Spatial Agency (ESA) and companies that supplies products to ESA: INSA, GMV, or CRISA.

Taken into account this industrial and technological context, three main categories of courses have been defined: graduate courses, doctorate courses and postgraduate courses.

Graduate courses focus on the preparation of professionals to be delivered to the society with sound enough knowledge to be productive in real work. In the context of our technical school (Telecommunications) and in particular in our specific area of interest (Telematic) the objective is to provide students with information about the existence and capabilities of current FDTs. This objective is covered by a last year course, that in the framework of signaling systems, management and planning of networks and integrated networks provides a brief presentation of the FDTs and focus on SDL and LOTOS.

Doctorate courses intend to provide deeper formation to students in specific field of new, relevant and state of the art technologies. Within this framework the variation in the contents of the courses, and the courses themselves change rapidly form one year to the next. During the first years of the FDTs definition and industrial diffusion, they became a “hot” technology in the field of protocols definition, design and implementations. Several groups of our university were forming research personnel in this field, so the number of courses related with FDTs were relatively high. During this period (1985-1990) several doctorates courses where granted: LOTOS language and Software engineering with LOTOS among others.

Today, a doctorate course uses FDTs as a tool for supporting Software Validation and Verification, Software Design, Object Oriented Software development, etc.

Finally, postgraduate courses (master course) cover the training of engineers that are actually working in companies, and that demand actualization on the last technologies. The rest of this paper is dedicated to describe a typical course granted during several years as a master course at DIT-UPM.

3 Didactic Material

It is usual that when you afford the problem of teaching a really new technology you do not have enough didactic material to complete a full course. But, only a few times in your life, you must prepare all the material from the scratch.

At the beginning, the material available was limited to: ISO8807 [7], CCS [10] book, and some material about ACTONE data Types [4]. Meanwhile this material may be sufficient for postgraduate course dedicated to doctoral students; it was clearly non-well suited for supporting courses for others students.

After the first experiences of teaching LOTOS courses with those basic didactic elements, the next material was clearly identified as mandatory for a correct course development:

- A reference book that provides a complete coverage of the language and its applications.
- Tutorials about LOTOS Behaviour and Data Types, that covers the gap between the basic mathematical bibliography and students background, mainly on Protocol and Computer Science.
- Syntactic and semantics analysis tools for supporting practices on basic language properties.
- Animation, simulation and compilation tools that allows to exercise, and the industrial application, of the high level language's capabilities.
- Illustrative application examples, which shows typical language structures and offers a collection of solutions to typical protocols and systems structured.

In order to support the development of the language itself, and with the purpose of facilitate the diffusion of the language, the next elements where development by different universities and research institutes of Europe.

3.1.1 Written material

The main support of any course is the written material. For supporting a complete course about LOTOS and their applications, it is required a reference book complemented with specific tutorials, and of course a good set of technical papers, for people that want to learn more about some specific field.

This kind of material was produced mainly between 87 and 90. In our case draft versions of our own material were used during the first courses:

- **Using Formal Description Techniques --- An Introduction to Estelle, LOTOS and SDL[24].-** Editor Kenneth J. Turner (1993). The basic reference book developed as a join effort of several European Universities and research institutions.
- **Introduction to the ISO Specification Language LOTOS [1].-** Tommaso Bolognesi and Ed Brinksma (1988). That provides a basic language introduction.
- **A Tutorial on LOTOS [3].-** Ed Brinksma (1985). Also a basic language tutorial.
- **A Tutorial on ADT Semantics for LOTOS Users; Part I: Fundamental Concepts [13]**.- José A. Mañas (1988). This 1st part presents, using a understandable language, the basic concepts of LOTOS ADTs.
- **A Tutorial on ADT Semantics for LOTOS Users; Part II: Operations on Types[14].-** José A. Mañas (1988). This second part covers the usage of ADTs, describing these ADTs from the point of view of a Programming language, as much it is possible, taken into account the specific characteristics of such ADT.

Complementary documentation about other FDTs may be found in many documents as:

- **An Introduction to Estelle: A Specification Language for Distributed Systems [2] .-** Stephan Budkowski and P. Dembinski (1987)
- **SDL -- CCITT Specification and Description Language [23].-** A. Rockfrom and R. Saracco (1988)

3.1.2 Tools

When teaching a language, in our case a formal language, the existence of tools is crucial to allow the realization of practices in order to permit the students work with the concepts described during the theory sessions.

In the case of formal language the possibilities are not restricted to compile and to execute a "program". So the next tools where developed, partially supported by international project, to provide a complete tools set for the language: syntactic and semantic analyzers and compiler tools (TOPO [15]), simulators (HIPPO [5]), transformation tools (LOLA[21]), which documentation is:

- **LOLA: Design and Verification of Protocols Using LOTOS [21].-** Juan Quemada, Angel Fernández and José A. Mañas (1988)
- **Transforming LOTOS Specifications with LOLA [22].-** The Parameterized Expansion.- Juan Quemada and Santiago Pavón and Angel Fernández (1988)
- **From LOTOS to C [17] .-** José A. Mañas and Tomás de Miguel (1988).
- **TOPO: Quick Reference -- C Code Generator [15].-** José A. Mañas, Tomás de Miguel, Tomás Robles, Joaquín Salvachúa and Gabriel Huecas and (1993)
- **Design of the LotoSphere Symbolic LOTOS Simulator [5] .-** Peter van Eijk and Henk Eertink (1990).

3.1.3 Application examples and guidelines

Language and tools where created at the same time, but at the beginning nobody knows exactly how to use the language and the coming tools in order to get the best results. The development of case studies in parallel with the language and the tools provided a good feedback and a bunch of example to illustrate the use of the most relevant characteristics of the language. Among others, we must refer to:

- **Abracadabra Protocol: Formal Description in LOTOS [12].-** José A. Mañas and Fernando Fournón (1993)
- **The Case of a Producer and a Consumer: An Example of Using the Sedos Compiler [11].-** José A. Mañas (1987)
- **Working Draft for the Guidelines for Application of Estelle, LOTOS and SDL [8].-** the abracadabra example (1988).
- **The Two-key system [20].-** Juan Quemada and Arturo Azcorra [1993]. It offers a complete example of the product life cycle.
- **The SRTS Experience: Using TOPO for LOTOS Design and Realization [18].-** Tomás de Miguel and Tomás Robles and Joaquín Salvachúa and Arturo Azcorra (1990)
- **Manual versus Hand Coded Implementation of LOTOS Data Types [16].-** José A. Mañas, Tomás de Miguel, Tomás Robles and Joaquín Salvachúa (1988)

4 Formal Methods course design

Many courses about LOTOS have been granted at DIT-UPM in the last decade. After some trials and the corresponding course modifications, we reached a structure that satisfies both professor and students.

The course combines theory and practice. Theory is divided into two blocks. The first one is devoted to the language elements. The second part described how the language and the different associated tools might support the life cycle development of a system.

The practices are divided in three blocks. The first one is for working with the basic language elements, to write simple and correct specifications. These practices use the front-end of TOPO. The second block of practices is dedicated to illustrate the symbolic execution, exploration of the states of a system, test of a specification, and in general to explore all the possibilities provided by the existence of

a formal description of the System under development. The last block is for working with the compiler tool (TOPO[15]), in order to produce prototypes, implementations and testing specifications and product.

5 Theoretical part

The approach to the LOTOS course is based on two main teaching blocks. The educational objective of the first block (language block) is the comprehension of the language itself. After completing this block a student should be capable of understanding and producing complex system specifications in LOTOS, selecting the most appropriate specification style. The educational objective of the second block (design methodology block) is the comprehension of how the language and the different associated tools can be used along the lifecycle of a system, mainly oriented to software implementations.

The existence of two formal components in LOTOS guided the organization of the language block in four sequential phases:

- **Introductory Concepts**: presents some fundamental concepts that are used in LOTOS
- **Basic LOTOS:** presents the control part (without data) of LOTOS and its supporting formal model.
- **ACT-ONE:** presents the abstract data type part of LOTOS.
- **Full LOTOS**: indicates how the behavioral and data components are combined to produce a complete system specification.

The students should acquire a level of LOTOS language comprehension before proceeding to the design methodology block. This block is itself structured in the following sequential phases:

- Uses of the Methodology: provides a brief introduction of the design process and a description of its fundamental design operation, which is the design step.
- The Design Process: provides an introduction to the three different design tasks, which must be fulfilled to perform a design step.
- Production of the Design: describes the design production task in detail. Implementation or Prototyping: describes the implementation task in detail.
- Assessment of the Design: describes the design assessment task in detail.
- The Two-Key System: provides a small example to illustrate the usage of the design methodology.

The following subsections provide a more detailed view of the contents of each of the educational phases belonging to the two blocks.

5.1 Introductory Concepts

Design languages were developed to allow designers to build models of (parts of) systems, for the purpose of analysis and design. LOTOS uses the concepts of process, event and behavior expression as basic architectural concepts for modeling distributed and concurrent systems. These concepts are introduced to the student to set the foundation of the further description of the language.

The student should finish this phase with a good comprehension of the meaning of observable behavior. The observable behavior of a system is described in LOTOS by means of a language element, in which the sequences of allowed events are represented. This language element is called *behavior expression*, which is a mathematical expression that receives a precise meaning in terms of the LOTOS semantic model.

Behavior expressions can be represented graphically as *behavior trees*. This representation is introduced to help visualizing the sequence of events and their dependencies, but its practical use is limited to simple cases of behavior. In behavior trees, branches are labeled with event names, and nodes represent states. The behavior is supposed to start from the top node. The occurrence of an

event selects a specific branch, and the behavior proceeds top-down, until a node that does not contain further outgoing branches is reached.

5.2 Basic LOTOS

LOTOS without ActOne is called Basic LOTOS. Experience shows that Basic LOTOS is easier to understand than Full LOTOS; furthermore Basic LOTOS can be generalized to Full LOTOS in a quite straightforward way. This phase begins by introducing the basic operators of LOTOS. These are the ones, which allow the representation of a finite and deterministic behavior: *inaction, action prefix* and *choice*.

Then, the internal event is introduced, analyzing its usage to describe non-deterministic behaviors. Finally, the remaining operators and constructions are introduced in order to cope with structuring, readability, and repetitive behavior representations. These operators and constructions are *process definition* and *Instantiation*, together with their usage to specify recursive behaviors, *successful termination*, *parallel composition*, *gate hiding, sequential composition* and *Disabling*.

5.3 ACT ONE

This phase begins by reminding the definition of an algebra, to continue with the different types of algebras with positive and negative constructs. Unicity of algebras is addressed introducing the concept of initial and final algebras.

Based on the mathematical description of initial algebras, the different ACT ONE constructs and operators are introduced. Emphasis is made on static semantics, because due to the overloading of identifiers this is an area that has been identified as particularly confusing to students.

5.4 Full LOTOS

In Full LOTOS an event is represented by a gate identifier and a list of interaction parameters, which are called *experiment offers*. There are two kinds of experiment offers: (i) *value offer* and *variable offer*. An event can only take place if there is a matching on the list of interaction parameters w.r.t. the sort of the parameters (considering their order) or with the values (if that is the case) in all behavior expressions participating in the event. Taking the possible combinations of value and variable offers for an event with a single experiment offer, it is possible to specify three different interaction types.

Very often we need to represent that the system will behave differently depending on certain conditions. These conditions may be either generated by past events or derived from some information available in the system or both. These conditions can be represented in LOTOS as operations of Boolean result involving process variables or interaction parameters. Conditional behavior can be represented in two ways: (i) restriction imposed before an event or (ii) restriction imposed at the occurrence of an event. The former is represented in LOTOS by *guards*, the latter by *selection predicate*.

Processes can be parameterized in LOTOS with data parameters. Process parameters are declared in process definitions as formal parameters; in process instantiations these parameters must be assigned to target values, i.e. value expressions that can be evaluated. As one could expect, it is required that the sorts of the target values in the process instantiations match those of the corresponding formal parameters in the process definition.

In Full LOTOS the concept of successful termination of behavior expressions is enriched with values that can be conveyed in the pseudo event ∂. Therefore the **exit** operators may contain an exit parameter list, which can be filled in by value expressions. A special syntax construct **any** is introduced to indicate that any value of a certain sort is allowed in the **exit**. Notice that the parameterless **exit** of Basic LOTOS is the special case of an empty exit parameter list.

Each valid LOTOS behavior expression has assigned to it a static property, which is called *functionality*. The functionality of a behavior expression is an ordered list of sorts, which indicates the exit parameter list associated with this behavior expression. In Full LOTOS we can extend sequential

composition by also allowing values to be passed between sequentially composed behavior expressions by using parameterized sequential composition.

Another important language construct is Local Value Definition. It provides a way to associate values to free variables in behavior expressions. This operator was introduced to allow more conciseness and better readability in specifications, by supporting the substitution of (eventually large) value expressions by a single identifier.

Choice of gates and choice of value constructs are basically shorthand notations to compact the specification text, and can in most of the cases be interpreted in terms of the choice and action prefix operators. The generalized parallel operator is also a shorthand notation defined to compact specifications and it is possible to interpret the generalized parallel operator in terms of parallel compositions of behavior expressions.

5.5 Design Methodology

This educational block provides an application oriented description of the methodology, together with the guidelines to be followed for its practical usage. The design methodology is a formally based stepwise refinement design approach. The methodology takes advantage of the design facilities provided by LOTOS. LOTOS is the formal description technique upon which the methodology is based and plays therefore a central role. Some previous knowledge of LOTOS and of its underlying theories is needed for the proper understanding of these guidelines.

The guidelines given and the activities considered are mainly of technical nature. They focus principally on the aspects related to the formalization of the design process. Links are provided such that this structure can be effectively connected into standard management, diagnostics or project control procedures. The basis for providing such links is the separation between design and assessment (or control) activities. The real application of the methodology should complement the guidelines given here with the particular management techniques of each user. Management techniques are different in most application environments.

The guidelines have been also conceived with enough flexibility, to allow its use with other existing design models, i.e. they are compatible with different models of software life cycles such as the waterfall model, rapid prototyping, or the spiral model. The methodology can be considered complementary to such life-cycle models. It gives guidance on how to formalize the design process in any of the models mentioned before.

The design methodology has been conceived to give formal support to an industrial design process, in order to achieve high quality designs. The usage of the formal description technique LOTOS as the basis of the design allows the use of this methodology with different design goals. Three main uses have been identified:

- **Production of a formal standard**: There exist applications ---open systems interconnection, portable O.S. interfaces,...--- in which a given standard must be generated before the product is really implemented on a large variety of different systems. A formal language like LOTOS is most suitable for expressing such a standard, eliminating the ambiguity and lack of conciseness of non-formal standards. One important feature of such a standard is its implementation independence.
- **Production of a product based on a standard:** Once a formal standard has been issued, the standard must be effectively implemented in many different systems.
- **Production of a product from scratch:** there are applications which have to produce a single type of product in which there is no need to produce or to adhere to an intermediate standard.

Although the three applications have many things in common, each one has to consider particular guidelines and design constraints. When necessary, particular guidelines will be given for each of the three cases.

The design process can be divided in four phases according to the goals existing in each part of it. The *design process* starts with a *requirements capturing* phase which will produce the requirements. The

requirements will constitute the central reference on which the design must be based. The requirements can be partially formalized with the help of LOTOS. The abstract nature of LOTOS allows the specification of implementation independent specifications.

Once the initial requirements are settled the design of the system starts by defining its architecture. This phase is called the *architectural phase*. An implementation is then produced with the architecture defined. This phase is called the *implementation phase*. The implementation is a representation of the system in a technology specific representation, which will be usually derived from the last LOTOS description of the system being defined. LOTOS provides formal support during these two phases, therefore this document provides only guidelines for performing the architectural and implementation phases.

A real system will be derived then from the implementation by transforming the technology specific description of the system into its physical realization. This phase is called the *realization phase*. The implementation and realization phases are treated as a unique phase in other methodologies.

The architectural and implementation phases consist of a sequence or a tree of design steps. The *design step* is the fundamental design operation, which produces a description of the system at a given level of abstraction.

The first design step produces the most abstract description of the system. Each of the following design steps derives a more complete or refined description of the system, which must be consistent with all the descriptions of the system produced in the previous refinements. The new description is said to be at a new *level of abstraction*. It is usually a representation of the system, which is a more complete model of the system being designed. The last design step should produce a complete model of the system, which must have a direct mapping into the implementation technology being used.

6 Laboratory and Practices

Three blocks of practices are interleaved with the theoretical sessions. The first one is for exercising the elements of the language. The second one for using symbolic tools and exploring the formal properties of the LOTOS specifications (using LOLA), and the third one for producing executable implementations (using TOPO).

The first block of practices uses the front-end of the TOPO compiler and is similar to the basic exercises we may do in any programming course, with the intention to familiarize students with the basic elements of the language. Blocks two and three are described in this section.

6.1 Learning LOTOS with LOLA

Practice is a fundamental basis to actually acquire real knowledge of any programming language. In the case of a specification language practice gets even more important as the abstraction level is higher, the concepts introduced in the theory classes are too many and too complex for the newcomer, and making effective usage of formal languages requires learning how to use tools that are complex and difficult to use effectively themselves.

The purpose of these practices is to show whom the formal support of the language help in the analysis of LOTOS specifications during the Design process.

One of the tools that help to understand better most concepts related to LOTOS behaviour semantics is LOLA. In outline, LOLA (LOtos LAboratory) is a tool that performs the following functionality set: symbolic execution, EFSM generation and testing [19]. This is the way we have carried out our work of teaching LOTOS with this application.

In the first place, it is paramount to devote a "blackboard session" to explain the concepts necessary to understand the tools for LOTOS. The students are given an introduction to the available toolset for LOTOS, from requirement capture to implementation. Functional decomposition, process architecture diagramming, step-by-step symbolic execution, debugging, EFSM computation, testing,

simulation with a time-extended model, implementation by annotation, conformance testing, are some of concepts reviewed to locate the tools to be used in the development cycle context. In our case, we tried to map the TOPO functionality set with the functionality set ideally desired in the perfect development environment model.

All LOTOS tools deal with specifications flattened. The students are warned about the gap between their original specification and the specification they will see in the development environment after flattening (a flat type definition, no process nesting, no identifier overload, etc). Second warning: equations are treated as rewrite rules. Therefore, students must be aware of termination and confluency by simple examples like x+y=y+x; and f(0)=1 and f(x)=x; fx(x); respectively. Similar considerations are made about non-guarded behaviours and states offering infinite actions.

Once these considerations have been made, there is a preliminary introduction to invoking LOLA from the TOPO toolset front-end and the list of commands available in LOLA. After this, a series of practical exercises are carried out in four sessions and organized as follows.

6.1.1 Session 1

1. Contact. Introduction to basic commands: help, print, stat, move, quit, tried on a simple buffer specification, as a first contact with the tool.
2. Checking syntax and semantics. Use a text editor to add recursion to a specification of a telephone set. The student gets in touch with the environment: editor and command structure of the TOPO toolset. LOTOS syntax, is faced for the first time and the student learns to understand the meaning of the lexical, syntax and static semantics error messages.
3. Step-by-step symbolic execution. The student acts as environment to choose among the different events offered by a specification. Undeterminism, environment, concurrency, event synchronization, expansion are concepts visited here. The user can see the way the system evolves through one-level expansions into a tree of actions.
4. Specification and debugging. The students are given an uninformal description of a system (usually a vendor machine) and they have to specify it from scratch. Action prefix, enabling, disabling operators and process instantiation arise naturally.

6.1.2 Session 2

5. Analysis. Case study of a system composed of a set of clients accessing to a transaction server by means of a bus. Practice of navigation and step-by-step execution to check that everything runs as expected. In this first version, the server is sequential: client processes have to wait for the server to complete any undergoing transaction before being served. Students are prompted to use advanced debugging features like watching the synchronization of actions that yield a global event and all its relevant information: the stacks of process instantiations involved in each execution thread, actions line numbers and value offers. The client-server specification has intentionally a deadlocking error that students have to detect and fix using functionality that displays the synchronization failed due to value offering mismatch.
6. Increasing concurrency. Students are requested to modify the client-server specification in order to allow for a given amount of concurrent transactions. Dynamic instantiation of concurrent processes and guarded behaviors.

6.1.3 Session 3

7. EFSM computation. After the students have learned about the Expansion Theorem, they have to practice with existing tools over several specifications. The example chosen to make them distinguish normal expansion and parameterized expansion[22] (actual process parameters are processed symbolically yielding a more compressed EFSM) are the Triangle of Pascal-

Tartaglia[9] and the Alternating Bit Protocol[22]. The state explosion problem arises quite naturally.

8. Testing. Students learn the testing concepts applied to LOTOS[19]: test types (accept and reject), response types (must, may pass) and tools available to implement this concept ranging from exhaustive analysis of specification-test composition (introducing the concept of verification) to random execution of traces (introducing validation). The students learn the different approaches to reduce the complexity of the test response computation, e.g. testing equivalent removal of internal actions, in the case of verification; and different resources to improve the quality of heuristic validation, e.g. bit state hashing[6].
   The example used is the specification of a resource-oriented buffer. The student is requested to try out several tests like input/output of messages in FIFO and LIFO fashion, and to print and analyze the tree of unsuccessful executions.

6.1.4 Session 4

9. Test suite design. Coming back to the client-server specifications (the concurrent and the serial server ones), the student will "hide" to the environment the access to the remote server and design a test suite to prove several properties: single access, three consecutive transactions, 3 clients served FIFO, LIFO and in any order, server [non-]concurrency, messages are delivered to the right client, etc. These tests must be processed in batch using the testing functionality of LOLA.
   Design of a protocol. The purpose of this practice is to show specific aspects involved in the specification of a protocol, basically the sort of problems that justify the existence of formal specification and validation and that are the gist of protocol engineering. The student is presented the Lynch protocol, a classical example that shows message duplication in a non-reliable channel illustrating the need for message numbering. Students are requested to specify a reliable semiduplex channel and test the protocol on it. After that, they have to specify a non-reliable channel and let them find out and enunciate why it fails, design three acceptance and rejection tests, and fix the protocol in order to get a reliable service. Finally, the students must convert protocol and line to full-duplex. This example serves as a hint of the complexity of designing a computer protocol.

6.2 Protocol implementation using TOPO

The student won't be able to get a full view of the whole life cycle of a formal specification with out experimenting the last parts of this cycle. This implies get a real working implementation and related it with the previous work done to get a sound specification of the system we are describing.

This skills are different from the previous acquired in the rest of the course. One previous knowledge that is supposed from the students is to be fluent in a system implementation programming language. This requirement was, sometimes, not meet by the students and implies that extra time has to be devoted to the explanation of this languages. This implies that some students missed most of the work done here.

One key point is to be able to go from the specification to the implementation and back to the specification. This implies the use of a tool instead of making a translation by hand, since this implies that ones the first implementation is done we must make parallel evolution the implementation, what sometimes is not easy.

The system programming language chosen was C. This is a very efficient language that allows the programmer to obtain most from the underlying facilities that the operating system can offered. This is very important in the case of protocol implementation since the performance extra gain for been more "close to the bone" is usually needed. Also is the most widely used language for protocol implementation in the industry so it's knowledge is good for the students.

The main objectives for this course are:

- The students will know the difference from a specification and an implementation of a protocol.
- The students will be able to take the decisions needed to get an implementation running.
- The students will be able to use a system implementation language in order to get the protocol running.

In order to reach these objectives we will have some blackboard classes where the students will receive some lessons about what and implementation means. That is:

- To close the open issues that may be present in the protocol.
- Mapping the abstract actions to concrete real ones (like really sending and receiving data) by means of side effects over LOTOS actions.
- Added real timing information.
- Avoiding non-determinism.

This can be done using the implementation facilities that are present inside the TOPO toolset. For this is critical the annotation facilities that are use for this purpouse.

The students are previously familiarized with the TOPO tools since the have use it for syntactic and semantic analysis, but in this course the more advanced facilities are presented in an incremental way.

6.2.1 Session 1

The goal for this session is that the students gets familiarized with the TOPO toolset and its facilities for getting an implementation from a specification; also the role of getting the specification annotated with implementation details that can no be expressed into the LOTOS specification language.

In order to have an incremental complexity approach for this session some pre-annotated specification is provided. The case of study chosen was the dinning philosophers.

This well-known specification is annotated to get some timing and some animation is provided on a screen for some of the actions that the run-time support performs.

The students get an implementation and see how the actions performed are related with the ones the see in the specification.

Also the are encouraged to change the annotation to see how the implementation stops working right. This way the know more about how the annotations works.

6.2.2 Session 2

The students are given a specification of a protocol not too complex like the abracadabra protocol [12]. The first goal is to use the annotations to give some timing and having the non-determinism that may be present in the specification removed. Also some implementation decisions are to be made.

We just give the students the specifications and some guidelines about how to annotate in the classes. Also some of the key points, like how to implement a timeout, was studied in this classes.

Ones the students know the specification the just annotated the specification for getting some traces of the events that are been performed and they know more about this protocol.

6.2.3 Session 3

Here the will get the protocol implementation. Since not all the students are fluent in C we simplify their task by giving some C code that implement more of the aspect related with the more operating system, mainly I/O procedures. With this the students implement the C functions that send and receive PDU and annotates the specification in order to use it.

For making some testing we set-up son serial connection between different PC in our laboratory so each student can run an instance of his implementation in the PC. Since if everything goes ok there are

some parts of the specification that are not exercised a switch was introduce in the cable connection between the PC's so the students can introduce error on the transmission.

In order to simply the structure of the laboratory some courses implement communication on top of TCP/IP. We must take into account that is for demonstration purposes (otherwise using a reliable protocol to implement a non-reliable protocol may look a bit silly). In this case the library offered to use sockets include a programmable random chance of a PDU not to be send or received.

Once the get an implementation the can use two instances to interoperate and see the protocol running.

6.2.4 Session 4

In this session the will use the annotations and the C code used in the last session with some traces that the obtained from the LOLA tool. In this way the get no an implementation for the specification but for a test of the specification.

Ones the get this implementation running the can interoperate with the implementations of the protocol that they obtained last session. This way the get a tester for the protocol.

Also the see if the annotation processes left out some properties that where present in the specification, like non determinism, but that is lost in the implementation process.

During these practical sessions, the students see really working the entire thing that they studied before and see who. The students can practice how an implementation is related with a specification and which is the way to get a real running implementation for a protocol.

7 Conclusions

The teaching of FDTs, and particularly LOTOS, has been a challenge for DIT-UPM. The first problems arise when we learned that the didactic material was not enough mature for teaching courses to students with a normal engineering background. So the first task, in parallel with the development of the language itself, was the creation of didactic material including written material, supporting tools, and reference examples. Tutorials, examples and tools like TOPO and LOLA where created consuming a lot of human effort.

The second step in this works was the consolidation of a course that covers the necessities of engineers that are working on the environment companies. This was achieved with the development of a full course that combines theory and practices focusing the problems from a practical point of view. This means that we reduced the amount of basic mathematics contents of the course in order to provide more information about the usage and practical application of the language.

The final conclusion of this experience is that hundred of engineers have attended such courses. They have improved their professional skills in the FDTs background, and the idea of the use of formal languages in the development of complex systems is now a common understanding of many companies of our near environment.

References

[1] Tommaso Bolognesi and Ed Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems.* 1988.

[2] Stephan Budkowski and P. Dembinski. An Introduction to Estelle: A Specification Language for Distributed Systems. Computer Networks and ISDN Systems. 1987.

[3] Ed. Brinksma. A Tutorial on LOTOS. *In Proceedings of the Protocol Specification, Testing, and Verification V.* 1985.

[4] H. Ehrig, W. Fey and H. Hansen. ACT ONE: An Algebraic Language with two Levels of Semantics. *Internal Report Tech Universitat Berlin*, 1983.

[5] Peter van Eijk and Henk Eertink. Design of the LotoSphere Symbolic LOTOS Simulator. *In Proceeding of the 3th International Conference on Formal Description Techniques*, Madrid, 1990.

[6] G.J. Holzmann. Design and Validation of Computer Protocols. Prentice Hall, 1991.

[7] ISO. Information Processing Systems -- Open Systems Interconnection -- LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. IS-8807. 1989.

[8] ISO. *Working Draft for the Guidelines for Application of Estelle, LOTOS and SDL.* ISO/IEC~JTC~1/SC~21/N2549. 1988

[9] D. Larrabeity. Ph.D. Tesis: "Contribución al Análisis del Espacio de Estados de Especificaciones LOTOS". 1996.

[10] Robin Milner. A Calculus of Communicating Systems. Serie: *Lecture Notes in Computer Science.* Spring-Verlang. 1980.

[11] J. A. Mañas. The Case of a Producer and a Consumer. An Example of Using the Sedos Compiler. Working paper, sedos/c3/wp/41/m. 1987.

[12] J. A. Mañas and Fernando Fournón. The Abracadabra Protocol. In Using Formal Description Techniques, Wiley, 1993.

[13] J. A. Mañas A Tutorial on ADT Semantics for LOTOS Users; Part I: Fundamental Concepts. Internal Report Dpt. Telematics Engineering Technical Univ. Madrid, 1998.

[14] J. A. Mañas. A Tutorial on ADT Semantics for LOTOS Users; Part II: Operations on Types. Internal Report Dpt. Telematics Engineering Technical Univ. Madrid, 1998.

[15] J. A. Mañas, T. de Miguel, T. Robles, J. Salvachúa, Gabriel Huecas and Marcelino Veiga. TOPO: Quick Reference -- C Code Generator. Internal Report of Dpt. Telematics Engineering Technical Univ. Madrid, 1993.

[16] J. A. Mañas, T. de Miguel, T. Robles and J. Salvachúa. Manual versus Hand Coded Implementation of LOTOS Data Types. Internal Report of Dpt. Telematics Engineering Technical Univ. Madrid, 1989.

[17] J. A. Mañas and T. de Miguel. From LOTOS to C. In Proceeding of the 1st International Conference on Formal Description Techniques, Stirling, 1988.

[18] T. de Miguel, T. Robles, J. Salvachúa and A. Azcorra. The SRTS Experience: Using TOPO for LOTOS Design and Realization. *In Proceeding of the 3th International Conference on Formal Description Techniques*, Madrid, 1990.

[19] R. de Nicola and M. Hennessy. *Testing Equivalences for Processes.* Theoretical Computer Science. 1984.

[20] J.Quemada and A. Azcorra. *The Two-key System.* In Using Formal Description Techniques, Wiley, 1993.

[21] J. Quemada, A. Fernández and J. A. Mañas. LOLA: Design and Verification of Protocols Using LOTOS. In Proceeding of the IBERIAN Conference on Data Communications. Lisbon 1987.

[22] J. Quemada, S. Pavón and A. Fernández. Transforming LOTOS Specifications with LOLA: The Parameterized Expanse. *In Proceeding of the 1st International Conference on Formal Description Techniques.* Stirling, 1988.

[23] A. Rockfrom and R. Saracco. SDL -- CCITT Specification and Description Language. IEE Natl. Comm. Conf. 1981.

[24] Kenneth J. Turner (Editor). Using Formal Description Techniques --- An Introduction to Estelle, LOTOS and SDL. Wiley, 1993.